Ebook

The Business Value of Reliability-Driven Software Delivery

Creating an Action Plan for Implementing Shift-Left Reliability

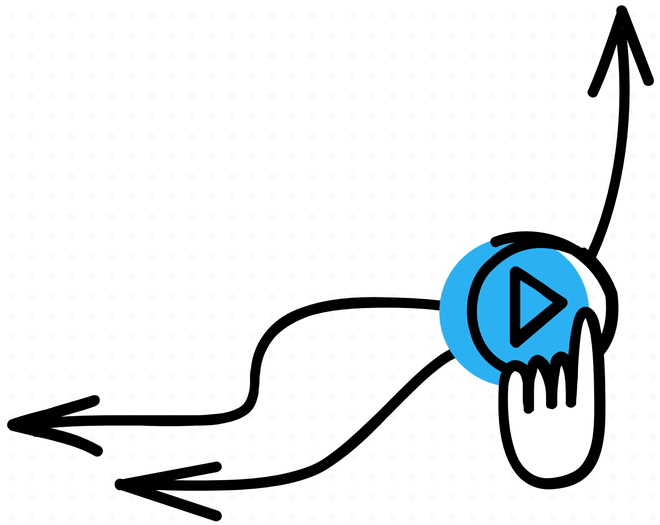
harness

Introduction to Shift-Left Reliability

As companies strive to gain a competitive advantage by delivering new software features faster, reliability of application services often suffers. These reliability issues manifest themselves in the form of SLA (Service Level Agreement) violations, customer complaints, low app store ratings, fewer customer purchases, and even loss of customers completely. To combat these negative consequences, businesses are grappling with the following question: "How do we deliver software at high velocity while making sure we achieve the best reliability?"

Many companies have realized that the answer to their question lies in Site Reliability Engineering practices–more specifically, with Service Level Objective (SLO) driven software delivery. SLOs can act as an early warning system, so application issues can be fixed before SLA violations occur and reputation or monetary damage is incurred.

SLOs are not a magic bullet, and the reality is that most companies are using SLOs in a reactive manner today. It's natural that in the early stages of adopting new processes, there will be opportunities for improvement as new ideas come about and new technologies are developed in support of those ideas.



Here are some characteristics of reactive SLO management:

- Developers are not working on reliability improvements during build and test phases of the SDLC (Software Delivery Lifecycle). Instead, they are re-tasked to perform reliability improvements after customers are impacted.
- Developers do not have access to SLO and error budget metrics to see how they are trending and take corrective action if needed. They only find out about SLO violations after they have become excessive.
- SREs or operations personnel manually reach out to developers when there are too many SLO violations or error budgets are drained. At this point, they instruct the developers to focus on reliability improvements instead of building new software features.
- Lack of automated guardrails in software delivery pipelines to automatically act upon SLO and error budget data. All actions resulting from SLO violations are manually initiated.

Companies don't have to live with this reactive methodology of SLO management. Taking a shift-left approach to reliability will amplify the benefits of SLO management practices by transitioning from reactive to proactive.

The principle of shift-left is to take a task that's traditionally done at a later stage of the software delivery process and **perform that task at earlier stages.**

The Business Consequences of Poor Reliability

Since the early days of IT, companies have searched for ways to proactively identify problems with their application services to avoid impact on customers. This led to the proliferation of observability, monitoring, and logging tools. These tools are good at detecting issues as they occur, but on their own, they cannot be used to determine how to make adjustments to the velocity of the software delivery practice to ensure that SLA violations don't occur.

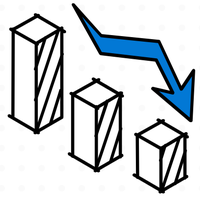
Here are some examples of potential business impacts determined through various studies:

Revenue Loss

ITIC's 2022 Global Server Hardware Security survey indicates that the Hourly Cost of Downtime exceeds $300,000 for 91% of SME and large enterprises. "Overall, 44% of mid-sized and large enterprise survey respondents reported that a single hour of downtime can potentially cost their businesses over one million ($1 million)."

SLA Penalties

Every software services company provides a list of penalties associated with SLA breaches. Typically the penalties increase as the severity of the SLA breach increases.



Customer Churn

A study conducted by Profitwell, a BI solutions provider, found that the cost of customer acquisition grew by 60% between 2014 and 2019. There are numerous other studies that support the idea that returning customers spend more than new customers and that retaining customers costs less than attracting new ones.

Defining Some Common Reliability Terms

There's a lot of jargon that comes with reliability. For one, people often confuse reliability with quality, when in fact, they are two very different challenges with different solutions. Here is a quick explainer of the terms you're likely to encounter around reliability.

Quality

Every company aims to build and deliver best-in-class software solutions that resonate with customer requirements. Whenever a software solution is ready to be shipped, the ultimate expectation criteria is quality. Software quality tells you how well a service performs its intended function. In other words, it's the conformance measure to the stated or implied product specification. Furthermore, you must define the right quality controls at each stage of the application delivery lifecycle to make sure that the product meets quality goals. Eventually, software quality will decrease bugs and errors while preventing costly rework.

Reliability

In simple terms, reliability is how well a product or service maintains its original quality over time and in various conditions. This is the fundamental feature of any product, and it's measured during or after a customer uses the product. For example, consider a scenario where a service has 90% reliability for a month. This means that under normal usage conditions during this month, there will be a 90% chance that the service won't experience any critical failures. This service attribute specifies how likely it will function without a failure for a given timeframe under predefined conditions.

In other words, reliability simply reflects how well a service behaves, and it is a means of measuring whether a service meets the expected behavior over time.

Service Level Objectives

A Service Level Objective (SLO) is a reliability target that is set to define the expected behavior of a service. In other words, an SLO is a target measure of how reliable a service is expected to be. For example, it could help you

determine the downtime, error rate, or service request response time that's acceptable for your service. But to understand what an SLO is, one must know what an Service Level Indicator is.

Service Level Indicators

A Service Level Indicator (SLI) is a metric that provides insights into the health of a service. It is the core metric used to indicate if an SLO is met. SLOs play a crucial role in shaping reliability goals that SREs must meet. They help site reliability engineers measure their success when accomplishing those goals by figuring out what and how to measure.

Service Level Agreements

Service Level Agreements (SLAs) are legal agreements that explain the implications if the service fails to meet the promised targets. For example, when a service has too many outages, driving availability below the promised level, the service provider may be subject to paying fines or penalties.

Error Budgets

An error budget is essentially an allowance for SLO violations that can accumulate over a certain timeframe for your service. It is the acceptable limit of unreliability before your customers are overly impacted. Failures are inevitable when you constantly change your systems. Therefore, normalizing failure as a part of the process helps teams balance innovation with the risk of SLA violation.

To improve the reliability and performance of your service, you must be capable of making important decisions, such as when and how much teams should prioritize new feature development work versus system stabilization efforts.

An error budget is a tool that helps teams take calculated risks and avoid obsessing over reliability. This tool helps the SRE and development teams work in tandem and control release velocity by ensuring that SLOs are met. Plenty of error budget remaining indicates that developers can work on new features without significant risk. Once the error budget is exhausted, teams should cease deploying new features and focus on service quality and reliability. Keeping tabs on the error budget consumption helps you determine the appropriate deployment rate for each engineering team.

Note: When calculating your Error Budget, you will need to decide if you want to filter out planned failure events (maintenance windows). If you want to get rid of planned maintenance windows then it might make sense to deduct that downtime from your error budget as a catalyst to achieve your goal. If no change to planned maintenance is desired then there is probably no point in docking the Error Budget during that time.

FLOWCHART

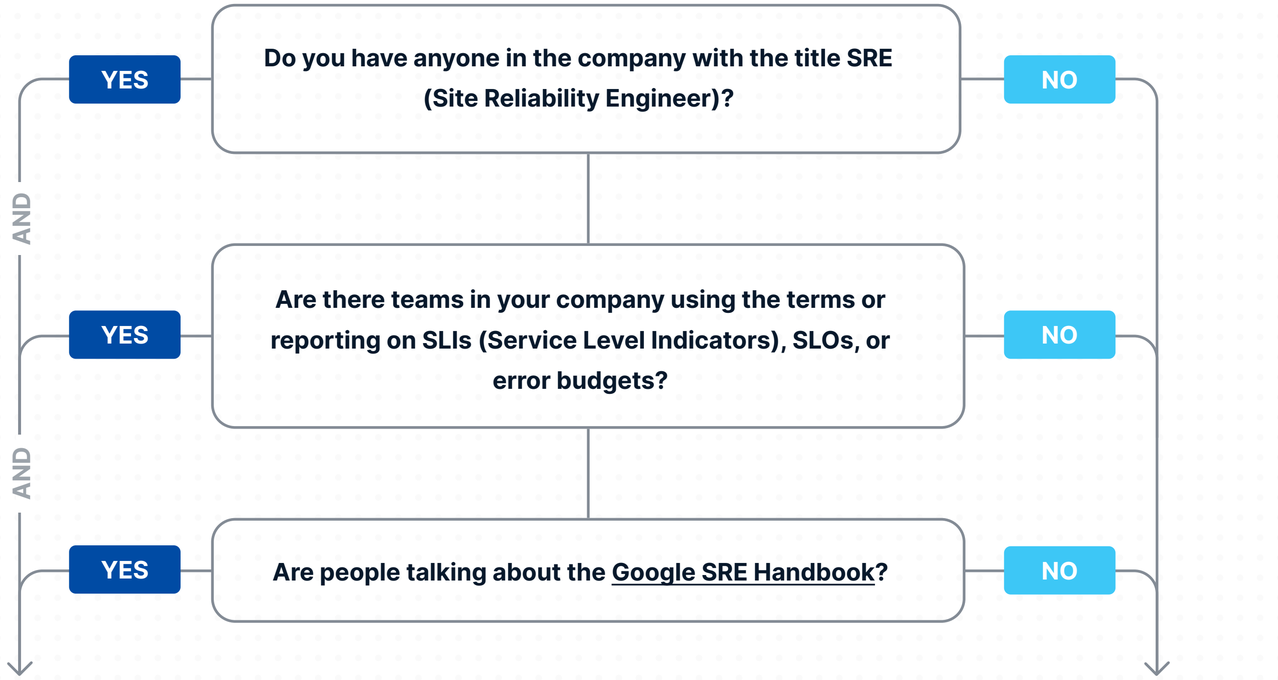
The Current State of Reliability-Driven Software Delivery

Most companies today are in 1 of 2 states when it comes to adopting SLO-driven software delivery:

- They have not started yet, but are interested in improving software reliability while maintaining or increasing software delivery velocity.
- They have already started down the path of SLO-driven software delivery, but they are bumping into challenges.

What State is Your Organization In?

Here are a few questions that can help you determine which state your company is in today:



You've probably already started down the path of SLOs.

Congratulations! Keep reading to discover how to advance your practice and overcome common challenges.

You haven't started down the SLO path just yet.

But that's okay—it's never too late to start. Take what you've learned to begin your SLO journey.

CHECKLIST

Overcoming Common Challenges of Reliability-Driven Software Delivery

Regardless of which state your company is currently in, it's important to understand the pitfalls that may be encountered while adopting SLO management practices. With the right combination of people, process, and technology, all of these challenges can be overcome.

Here's a checklist of the most common challenges to set strategy in place to overcome when adopting reliability-driven software delivery.

It's important to note that **all of these challenges can be avoided or overcome.**

- **Building an SLO management practice takes time** – It takes a combination of the right people, with the right knowledge to build the practice of SLO management. This often requires hiring new employees and giving them time to change the processes, tools, and culture within the organization.

- **The reliability and engineering teams might not be working from the same data** – The reliability team keeps an eye on SLOs while the engineering team is heads-down building software. When reliability issues occur, the engineering team is surprised when they are asked to slow down.

- **Reliability teams burn valuable time manually managing, tracking, and taking actions on SLOs** – Not only does this cause extra work for reliability engineers, but it leads to errors and inconsistent governance of software delivery pipelines.

- **Determining what changes have impacted SLOs can be difficult and slow** – When SLOs are breached, it's crucial to identify the root cause and remediate it before SLAs are violated and penalties are incurred. With frequent changes, this can be a difficult and time-consuming task.

- **Scaling SLO management beyond a handful of application services can be cumbersome** – With so much manual process, it becomes difficult to adopt SLO management across all services that need it.

- **Overall reliability of applications might improve, but too slowly** – Even with manual processes, reliability could improve over time. Without tooling designed to accelerate this process, improvements will be slow.

- **Verifying the quality and reliability of individual deployments is often a manual process** – After each software deployment, engineers look at logs and metrics for hours to determine the quality of the software. Reliability engineers look at similar dashboards for days or even weeks to determine the reliability of each deployment.

Keep reading to learn more about how all of these challenges can be overcome.

IMPROVEMENTS

The Business Value of Reliability-Driven Software Delivery

There's a reason companies either put up with or overcome the challenges listed above when adopting reliability-driven software delivery. It's because the potential business value is significant. SLOs and error budgets will help your technical teams avoid excessive customer impact.

Continuous reliability improvements will make your customers even happier since the software will function as expected, when needed. When the application services improve, your developers can focus on delivering new business functionality and that makes them happier too. All of this leads to:

Better customer retention

Most companies agree that it costs more to attract new customers than to retain existing customers. There's some disagreement about how much the cost differential is but it's generally agreed that it costs somewhere between 3-5 times as much to attract those new customers as retaining your existing customers. Keeping the ones you already have is just good business.

Higher revenue

Happy customers spend more. A study by Harvard Business Review found that "customers who had the best past experiences spend 140% more compared to those who had the poorest past experience".

Faster innovation cycles

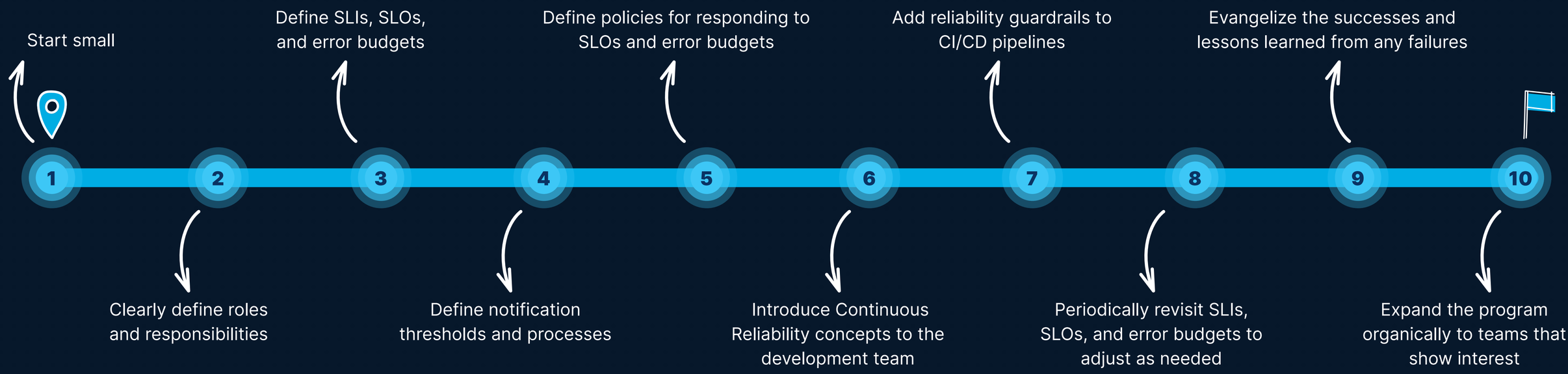
When software is highly reliable, developers can focus on writing new business functionality instead of optimizing and fixing pre-existing code. It has become commonly accepted that software innovation creates a competitive advantage. Innovating rapidly and have highly reliable software is a "best of both worlds" scenario that is achievable using reliability-driven software delivery.

Better employee retention

Nobody likes to be in constant firefighting mode or woken up in the middle of the night to fix broken application services. Developers want to create new features and functionality, but they can't do that if they're constantly prioritizing bug fixes. When software is plagued with reliability issues, developers will only invest so much time and effort into stabilization before they start looking for employment elsewhere. This situation can be avoided by promoting a culture where reliability practices are part of the entire SDLC. When this occurs, application services encounter few reliability issues, developers are free to work on creating new functionality, and employee churn rates are low.

A Roadmap for Adopting Reliability-Driven Software Delivery

Getting started with reliability-driven software delivery doesn't have to be painful and slow. You might choose to hire someone with the right experience or you might want to run an experiment using your existing DevOps and engineering staff. Either way, here are suggested steps to help make your successful when implementing reliability-driven software delivery.



1. **Start small**

   Choose a single application service to focus on and gain experience.

2. **Clearly define roles and responsibilities**

   Who will perform all of the various functions associated with the steps in this roadmap? Who will act as the program manager to coordinate activities? Who will be responsible for reporting to executive staff? Who wil provide budget for personnel and tooling? All of these questions need to be answered and accounted for in order to successfully implement this roadmap for reliability-driven software delivery.

3. **Define SLIs, SLOs, and error budgets**

   This should be a collaborative effort between those responsible for reliability (SRE, DevOps, Ops) and the development team responsible for the service. Start at the overall application boundary before applying SLO management to individual application services.

4. **Define notification thresholds and processes**

   When SLO's are breached and error budgets are depleted, the proper teams need to know about it so they can do something about it. Ideally, these notifications will provide some advanced warning of impending doom so that the proper teams can take proactive steps to avoid negative customer impacts like outages or excessive slowness.

The proper teams should be able to take proactive steps **to avoid negative customer impacts, like outages or excessive slowness.**

5 Define policies for responding to SLOs and error budgets

The team responsible for responding to SLO and error budget notifications MUST be granted authority to act as required. Clearly define what actions this team can take to make a meaningful impact and make sure there is an escalation process in place for any disputes that arise as a result. Ensure that all teams are aware of these authorizations and processes.

6 Introduce Continuous Reliability concepts to the development team

Continuous Reliability concepts include activities like exception identification and debugging, automated deployment verification, fault injection testing, and chaos engineering. All of these activities can, and should, be performed at various stages of the software delivery process.

7 Add reliability guardrails to CI/CD pipelines

Reliability guardrails are a way to automatically control the software delivery process based upon the status of SLOs and error budgets. These guardrails are programmatic steps within delivery pipelines that determine whether or not deployments can proceed. The reliability engineers and developers can collaboratively determine what guardrails to put in place at what thresholds.

8 Periodically revisit SLIs, SLOs, and error budgets to adjust as needed

Since every application service can have different tolerances for risk and different needs for feature velocity, it's natural that the engineering and reliability teams will need to revisit the metrics and thresholds used with SLIs, SLOs, and error budgets. Define some periodic review timeframes at the outset of the project.

9 Evangelize the successes and lessons learned from any failures

In IT, we've learned that even our failures can be viewed as successes as long as we are rapidly learning and adapting. Document every success and learn from every failure. Document how you will turn past failures into future successes and spread the word liberally when you have meaningful success.

10 Expand the program organically to teams that show interest

The goal of your evangelism effort is to help other teams gain the confidence to change their culture (ultimately for the overall benefit of the business and consumer). The more teams that adopt reliability-driven software delivery processes, the larger the potential benefit to the business.

The more teams that adopt reliability-driven software delivery processes, **the larger the potential benefit to the business.**

Start Your Reliability-Driven Software Delivery Initiative Today

As companies look to gain competitive advantage by delivering new software features faster, they find that reliability is often compromised by moving so quickly. As a result, they have started to adopt the SRE practices of SLO management and chaos engineering, but they have faced new challenges related to these practices. They've also realized that using SLO management in a reactive manner does not meet their reliability needs, leaving them open to SLA violations and other negative consequences.

Moving to a shift-left reliability model like reliability-driven software delivery is proven to minimize the risk of SLA violations while maintaining the competitive advantage of high velocity feature delivery. By using the right processes and tooling, reliability-driven software delivery can consistently scale to meet the demands of any size organization.

By using the right processes and tooling, reliability-driven software delivery can consistently scale to meet the demands of any size organization.

How Harness Can Help

The Harness Platform enables businesses to manage deployment by tracking key software reliability metrics. Engineering organizations can accelerate or slow down software releases based on leading indicators from the Harness Reliability Dashboard.

The Harness Platform's Improve Reliability Solution encompasses two standalone product modules: Service Reliability Management (SRM) and Chaos Engineering (CE). While SRM enables businesses to measure, understand, and improve the health of their application services, Harness CE helps businesses proactively address system failures that cause unplanned downtime before the failure happens in production.

Harness CE empowers enterprises to move fast while maturing the reliability of their systems and team. Chaos Engineering steps beyond traditional testing by combining deployments with the infrastructure they run on, and demonstrating how systems will respond to real-world failure scenarios. With Harness CE, engineering organizations get the support, onboarding, and expertise needed to quickly scale chaos engineering practices.

Harness SRM enables proactive reliability-driven software delivery for any organization, whether just getting started down the path of SLO management or at any experience level beyond that. SRM helps companies adopt, scale, and automate SLO management, so they can reap the benefits of improved reliability without suffering the common pain points.

Ultimately, Leveraging SRM and CE for reliability-driven software delivery ensures systems maintain reliability through failures, contractual obligations are met, and resilience mechanisms are effective.

With Harness, you're not just measuring reliability; you're also improving it.

Explore how Harness can help your organization improve reliability with a free discovery session to discuss your business, priorities, and challenges.

Schedule a Discovery Session

The Modern Software Delivery Platform™

Follow us on

/harnessio

/harnessinc

Contact us on

www.harness.io