(Un)Reliability Budgets

Finding Balance between Innovation and Reliability

MARK D. ROTH

Mark Roth has been a Site Reliability Engineer at Google's Mountain View office for over a decade. He has worked on a variety of projects, including Gmail, Google Accounts, Monitoring Infrastructure, and Compute Resource Management. Before coming to Google, he managed production UNIX systems at the University of Illinois at Urbana-Champaign, where he authored a number of open-source software packages. roth@google.com

Google is constantly changing our software to implement new, useful features for our users. Unfortunately, making changes is inherently risky. Google services are quite complex, and any new feature might accidentally cause problems for users. In fact, most outages of Google services are the result of deploying a change. As a consequence, there is an inherent tension between the desire to innovate quickly and to keep the site reliable. Google manages this tension by using a metrics-based approach called an unreliability budget, which provides an objective metric to guide decisions involving tradeoffs between innovation and reliability.

Structural Tension

The tension between innovation and change is reflected most strongly in the relationship between the SRE team and the corresponding Product Development team for any given application. This is partly due to the inherent conflict between the two teams' goals. Product Development's performance is largely evaluated based on product velocity, so they have incentive to get new code out as quickly as possible. However, SRE's performance is evaluated based on how reliable the service is, which means they are generally motivated to push back against a high rate of change. In addition, there is information asymmetry between the two teams. The product developers have more visibility into the time and effort involved in writing and releasing their code, while the SREs have more visibility into the service's reliability.

This inherent structural tension between Product Development and SRE manifests itself in disagreements in a number of areas where it is important to find the right balance between innovation and change. Here are some of the areas:

Software Fault Tolerance. When writing software, it's important to anticipate the possible failure modes and ensure that the software will handle them. However, there are an almost infinite number of ways in which software can fail, and product developers do not have an infinite amount of time to address those cases. Spending too little time on this results in brittle software, thus increasing outages; spending too much time on this means that it takes longer to finish the software, thus decreasing innovation. What is the right balance?

Testing. Too little testing results in bad, unreliable software. Too much testing can delay the software from ever being released and increase ongoing code maintenance costs due to the additional tests. Google product developers have many software testing tools at their disposal, but how much testing is enough?

Push Frequency. Some teams prefer to push a new software release monthly or weekly. Others would rather push daily or multiple times each day. Even if a push is mostly automated, it may still require work on the part of the SREs. Each push is risky. A bad push can result in a user-visible outage. Even without a user-visible outage, there may be a reduction in reliability during the push due to the fact that while some systems are upgraded, the others take on the additional load, possibly affecting latency. What's the best frequency for the application?

Canary Duration and Size. When pushing new software, most teams first push to a small subset of the total number of deployed instances, so that if there is a problem, it will only affect a subset of users. This is referred to as a "canary," named after the practice of using a canary to detect carbon monoxide in coal mines. Only after the code is deemed stable for some period of time in canary will it be pushed out to the rest of production. But how long should a change be canaried before it is deemed safe for the rest of production? Too little time and we risk not catching problems before they go to the rest of production; too much time and we decrease the rate at which changes can be deployed. Also, how large of a subset should the canary be? Too small and we risk not having a large enough sample size to catch problems before they go to production; too large and we risk any potential problems causing too large of an impact before they are caught. What is the right balance for the application?

Push Retry Methods. Sometimes a bad push is discovered and the service is reverted to the previous release. When this happens there is a temptation to make a quick fix and try again. Often these quick fixes are not as well tested, and the risk is increased. Alternatively, some groups prefer to wait for the next push cycle, whether weekly or daily. We find that both methods result in the same rate of new features making it into production, but the former method results in many more pushes and reverted bad pushes, which creates work and stress for the SREs. Is it better to fix something quickly or do a full suite of tests?

The two teams need to negotiate to find the right balance in these areas. However, we don't want this negotiation to be driven based on the negotiating skills of the engineers involved. We also don't want this to be decided by politics, personal feelings, or just plain hope. (Indeed, SRE's unofficial motto is "Hope is not a strategy.") Instead, we want an objective metric, agreed upon by both sides, that can be used to guide the negotiations in a reproducible way. Google is a data-driven company, and we want the decision to be based on hard data.

Unreliability Budgets

For these decisions to be made based on objective data, the two teams jointly define a quarterly unreliability budget based on the service's SLO (service level objective, or the goal of how reliable a service should be). The unreliability budget provides a clear, objective metric that determines how unreliable the service is allowed to be within a single quarter. This takes the politics out of the negotiation between the SREs and the product developers when deciding how much risk to allow.

The unreliability budget works as follows: Product Management sets a "Quarterly SLO goal," which sets an expectation of how much uptime the service should have. The actual uptime is measured by a neutral third party, our monitoring system. The difference between these two numbers is the "budget" of how much "unreliability" is remaining for the quarter. As long as the uptime measured is above the SLO, new releases can be pushed.

As a hypothetical example, let's imagine that a service's SLO is that it will successfully serve 99.999% of all queries. This means that the service's unreliability budget is that it can fail 0.001% of the time within a given quarter. So if a given problem causes us to fail 0.0002% of the expected queries for the quarter, we would consider that it used up 20% of the service's unreliability budget for the quarter. Once the unreliability budget for the quarter has been spent, no more changes will be deployed (other than critical bug fixes), since they could cause unreliability that the service can't afford.

The actual SLO for a given application may actually be a much more complicated calculation involving latency, data freshness, and other factors. In some cases, a successful push may reduce the SLO slightly even though no downtime is visible to the users. For example, while some servers are being upgraded, others take on the extra traffic, and thus latency may increase.

Benefits

The main benefit of an unreliability budget is that it provides a common incentive that allows both Product Development and SRE to focus on finding the right balance between innovation and reliability.

For example, if Product Development wants to skimp on testing or increase push velocity and SRE is resistant, the unreliability budget guides the decision. When the budget is big, the product developers can take more risks. When the budget is nearly drained, the product developers themselves will push for more testing or slower push velocity, because they don't want to risk using up the budget and stall their launch. In effect, the Product Development team becomes self-policing. They know the budget and can manage their own risk.

The unreliability budget also largely eliminates tension between Product Development and SRE, because SRE no longer needs to be in the position of making subjective judgment calls on individual push requests from product developers or adopting blanket and increasingly arbitrary rules such as "new releases are pushed if and only if Product Development wins a game of fizzbin when the moon is full" [1] in an attempt to prevent repetition of previously encountered outages. Instead, SRE just needs to measure and enforce the agreed upon unreliability budget. If they need to say no, they can point at an objective metric that Product Development has already agreed to and cannot argue with. Thus, instead of viewing SRE as an obstacle, the Product Development team partners closely with SRE on ensuring appropriate velocity/reliability tradeoffs.

What happens if a network outage or datacenter failure reduces the measured SLO? Yes, events like that consume the budget, too. As a result, the number of new pushes may be reduced for the remainder of the quarter. The entire team is okay with this because everyone shares the responsibility for uptime. No one person is to blame for such an incident. On the other hand, Google has mechanisms to "route around" such outages so they are invisible to our users. If such an event actually does affect the service, the team can focus on improving their use of the redundancy and failover mechanisms rather than waste time finger-pointing.

Finally, because the unreliability budget is defined in terms of the application's SLO, it also helps to highlight some of the costs of overly high reliability targets, in terms of both inflexibility and slow innovation. If the team is having trouble getting new features out, then they may elect to loosen the SLO (thus increasing the unreliability budget) in order to increase innovation. At Google, doing a little better than the SLO is good, but exceeding it greatly is not considered something to be proud of; instead, it is an indication that the team is not taking enough risks or the service is over-provisioned. Google encourages smart risk-taking to increase innovation, and the unreliability budget helps us make sure that we're doing that.

Conclusion

When two groups work as a team and share responsibility for the uptime of a service, it is important to have a neutral, non-political way to guide decisions of balance. Whether it is how much testing is enough, how often to push, or how to recover from failed pushes, these are not easy decisions to make. While product developers are under pressure to advance their products rapidly and SREs are always mindful of stability, the unreliability budget gives the team a neutral, non-political, and data-driven way to find balance in all these areas and more. The result is a team that works better together and more effectively.

Acknowledgments

Thank you to Tom Limoncelli, now at Stack Exchange, Inc., for contributing to an early draft, Dave O'Connor for his invaluable comments, and Carmela Quinito for editorial review of this article.

Reference

[1] Fizzbin: http://www.imdb.com/title/tt0708412/quotes.

Publish and Present Your Work at USENIX Conferences

The program committees of the following conferences are seeking submissions. CiteSeer ranks the USENIX Conference Proceedings among the top ten highest-impact publication venues for computer science.

Get more details about these Calls at **www.usenix.org/cfp.**

JESA: Journal of Education in System Administration

Submissions due: August 14, 2015
www.usenix.org/jesa/cfp

USENIX is proud to announce the creation of a new *Journal of Education in System Administration (JESA). JESA* brings together researchers, educators and experts from a variety of disciplines, ranging from informatics, information technology, computer science, networking, system administration, security and pedagogics. *JESA* seeks to publish original research on important problems in all aspects of education in system administration. The mission of *JESA* is therefore to be a body of peer-reviewed, high-quality work addressing the challenges in system administration education.

URES '15: 2015 USENIX Release Engineering Summit

November 13, 2015, Washington, D.C.
Submissions due: September 4, 2015
www.usenix.org/ures15/cfp

At the third USENIX Release Engineering Summit (URES '15), members of the release engineering community will come together to advance the state of release engineering, discuss its problems and solutions, and provide a forum for communication for members of this quickly growing field. URES '15 is looking for relevant and engaging speakers for our event on November 13, 2015, in Washington, D.C. We are excited that this year LISA attendees will be able to drop in on talks so we expect a large audience.

URES brings together people from all areas of release engineering—release engineers, developers, managers, site reliability engineers and others—to identify and help propose solutions for the most difficult problems in release engineering today.

NSDI '16: 13th USENIX Symposium on Networked Systems Design and Implementation

March 16-18, 2016, Santa Clara, CA
Paper titles and abstracts due: September 17, 2015
Complete paper submissions due: September 24, 2015
www.usenix.org/nsdi16/cfp

The 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) will focus on the design principles, implementation, and practical evaluation of networked and distributed systems. Our goal is to bring together researchers from across the networking and systems community to foster a broad approach to addressing overlapping research challenges.

NSDI provides a high quality, single-track forum for presenting results and discussing ideas that further the knowledge and understanding of the networked systems community as a whole, continue a significant research dialog, or push the architectural boundaries of network services.

FAST '16: 14th USENIX Conference on File and Storage Technologies

February 22-25, 2016, Santa Clara, CA
Submissions due: September 21, 2015
www.usenix.org/fast16/cfp

The 14th USENIX Conference on File and Storage Technologies (FAST '16) brings together storage-system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems. The program committee will interpret "storage systems" broadly; everything from low-level storage devices to information management is of interest. The conference will consist of technical presentations including refereed papers, Work-in-Progress (WiP) reports, poster sessions, and tutorials.

www.usenix.org/cfp