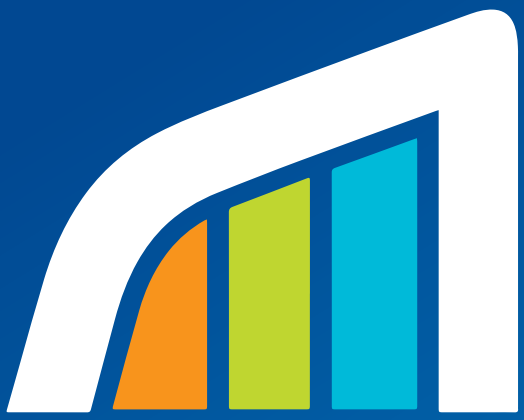


Rollbar

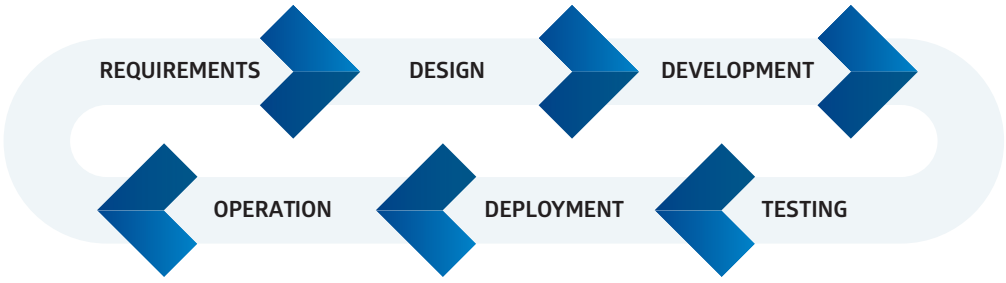
The Case For Monitoring Errors Earlier In The Software Development Life Cycle

Mature production monitoring and observability practices have been long considered a key component of the DevOps movement. In this book, we'll explore the emerging use case of implementing error monitoring and reporting as an observability practice to pre-production environments earlier in the Software Development Life Cycle (SDLC). We'll also explore how technical engineering organizations building production applications in the cloud, from lean startups to mature organizations, are capitalizing on these benefits today.

THE SDLC EXPLAINED

The SDLC, or Software Development Life Cycle, is the process through which technical organizations build, test, and ultimately deliver software to their customers. For SaaS organizations, the software is hosted and operated by the same organization building the system, making maintenance and operation of the system a key component of the SDLC.

Depending on the kind of products being built by an organization, there are plenty of references to various SDLC flows that include more or fewer phases, but for the purpose of this paper, we'll focus on the following set of phases:



The one and only purpose of the SDLC is to deliver value to the software's users through features, fixes, and improvements. As these new features, fixes, and improvements are ideated, built, and shipped, the quality of the software shipped becomes the pivotal factor for building trust with users.

RELYING ON DEDICATED ENVIRONMENTS

With over a decade of architecture and tooling evolution, the era of microservices and distributed systems has arrived. These days, expectations of availability and response time are high, and the complexity of operating such software has increased dramatically. Gone are the days of monolith architectures, run-of-the-mill relational databases, and singleton load balancers.

Alongside this progression on system complexity, teams in charge of operating this software have had to resort to dedicated environments to better replicate production in all phases of the SDLC. An environment can be seen as a self-contained, functional instance of the system being built. This representation goes all the way from the low-level infrastructure to the high-level software running on top.

The most common SDLC environments that organizations build and operate are as follows:

Environment	Dev	QA	Staging	Production
Purpose	Development of new features	Integration of the system's components and testing of new features	Acceptance testing of the full system with production-like data and infrastructure	Serving the end users
Priority	Cost	Stability	Production resemblance	Availability and performance
Replicas	Usually 1 per developer	Usually 1 per concurrent CI build being run	Usually 1	Usually 1
Duration	Usually ephemeral	Sometimes ephemeral	Rarely ephemeral	Rarely ephemeral

Dedicated environments replicating complex systems are inherently hard to operate and understand. Debugging when issues happen becomes a big challenge for organizations. As a result, these organizations build dedicated teams and practices that resemble those of production operations to handle this complexity.

As an environment grows in complexity and accumulates state, the effort required to keep its stability and behavior predictable increases. Environment re-use and sharing makes this problem even more difficult.

COMMON SIGNS THAT THE COMPLEXITY OF YOUR ENVIRONMENT IS IMPACTING YOUR ABILITY TO DELIVER SOFTWARE:

1. Teams struggle to coordinate the use of QA or staging environments
2. Continuous Integration builds fail more often than they pass
3. Engineers ignore failed builds and re-run tests to get a pass
4. Bugs caught in CI builds take hours and sometimes days to resolve
5. Deployments are done sporadically, off hours, and require all hands on deck

COMMON PRACTICES ORGANIZATIONS USE TO TACKLE THIS:

1. Building dedicated engineering services and QA teams in charge of improving the SDLC
2. Centralized logging for QA and staging environments
3. Performance metric collection and dashboarding for QA and staging environments
4. Configuration management and scheduled cleanup/re-building of environments
5. Ephemeral QA environments that remain clean at all times and are used only once

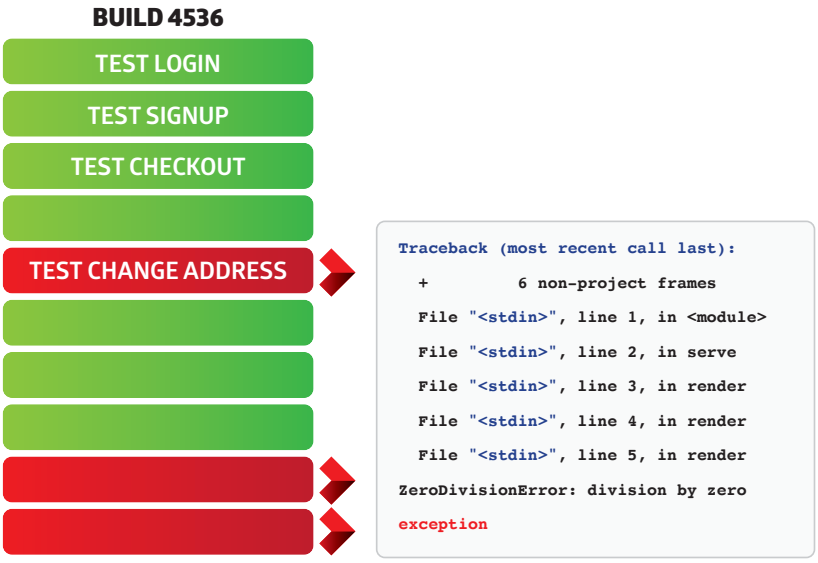
OBSERVE EARLY, OBSERVE OFTEN WITH ERROR MONITORING

As a key set of practices used to manage production environments, observability has been coined as the umbrella term that groups the observation of the system's behavior in real time, to better understand what's happening across the system's components and to the environment as a whole.

Over time, new and more advanced practices have been introduced to improve observability, but some of the most common include:

- Centralized logging
- Performance and functional metrics
- Functional testing
- Error monitoring

Through error monitoring, software code exceptions are intercepted, collected, and analyzed remotely to gain better insight into unexpected events happening inside the system. Through this practice, thousands of exceptions are grouped and turned into a manageable set of insights for engineering organizations to handle. As the impact of some exceptions will be directly visible to users, while others not so much, collecting analyzing their impact off-band can deliver a substantial increase in stability and reduce debugging time.



With the advent of CI, automated testing, and the constant pursuit of test coverage from the get-go, error monitoring can deliver a great deal of value during all phases of the SDLC, not just during production operation, as it has been commonly used up to this point. This move "up the cycle" of observability practices has been happening for almost a decade, but it's the first time that common practices and available technologies have allowed error monitoring to follow suit.

BENEFITS OF ERROR MONITORING EARLIER IN THE SDLC

The benefits we've observed of running error monitoring at earlier stages in the SLDC are:

- Faster resolution of build failures
- De-duplication of bug reports
- Improved bug reporting and communication
- Correlation of errors with previous occurrences
- Checking the effectiveness of the QA process

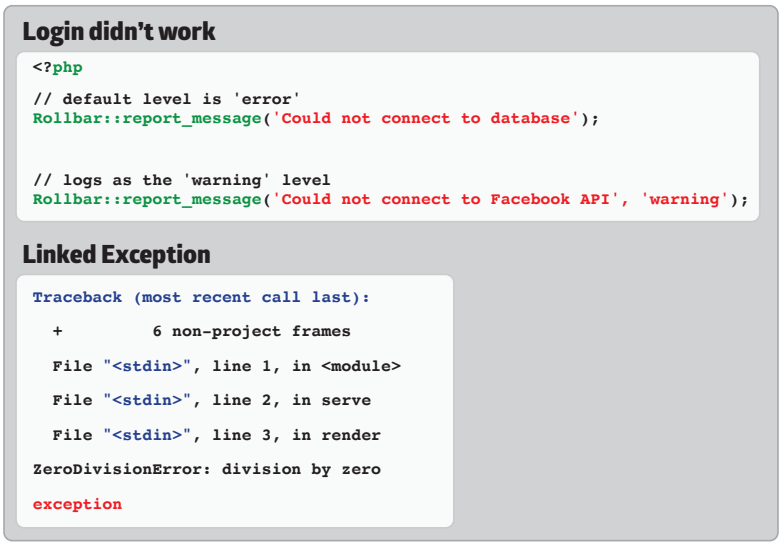
Let's address these in more detail.

FASTER RESOLUTION OF BUILD FAILURES

As we earlier concluded, investigation of build failures and bugs is becoming a greater challenge and a more pivotal piece of the SDLC. With multiple microservices and distributed heterogeneous systems, failures tend to be influenced by the timing and coordination of many small events. Sometimes these individual events are investigated by different individuals or even teams. When code exceptions and the values of all local variables (captured automatically) at the time of an error can be retrieved quickly and across the organization, the effort needed to investigate and coordinate a resolution is greatly reduced.

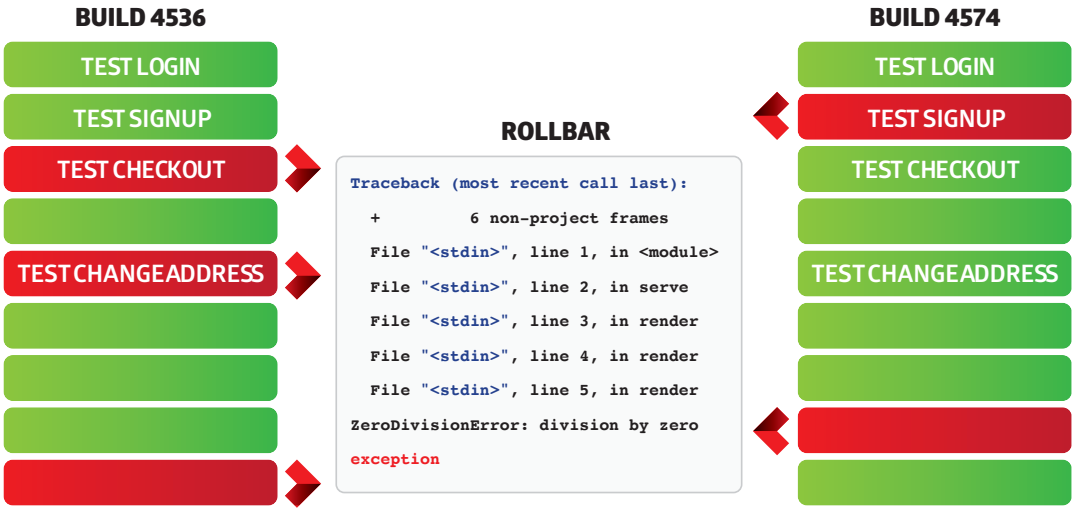
DE-DUPLICATION OF BUG REPORTS

With the advent of APIs and code reuse across distributed systems, a common problem that arises is that a single bug in a microservice can lead to dozens of slightly different failures in functionality across the whole system. In an organization where code coverage is pursued and the number of tests is high, a single line of code can trigger failures in dozens, if not hundreds, of tests, escalating to many hours of individual triaging and investigation to make sure everything is resolved before a new build is attempted. Being able to trace code exceptions to the original failure and to group these test failures for a single investigation can dramatically reduce the amount of work spent in development.



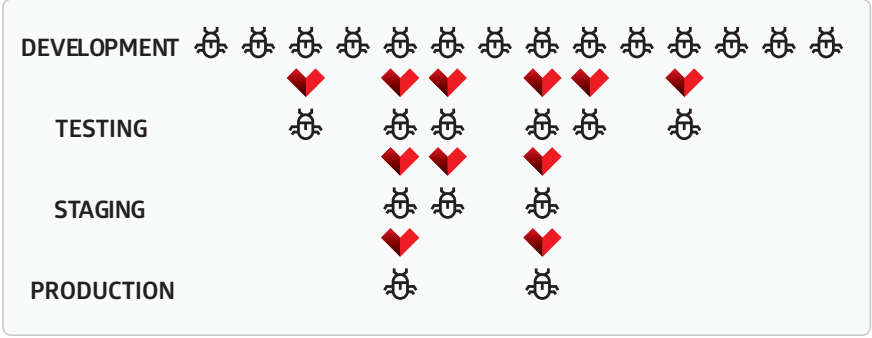
IMPROVED BUG REPORTING AND COMMUNICATION

When the team responsible for collecting and reporting bugs is different from the team fixing the code, the issue described above can take on a social character and the tension between these teams can grow quickly. The lack of insight and knowledge in the reporting side will frustrate developers, while the lack of visibility and foresight on the development side will frustrate the team responsible for quality. Exception details and linking of bug reports based on exceptions can greatly reduce friction by enhancing the quality of bug reports and acting as a connector between code and functionality, as well as a common ground between both teams. Additionally, resolution of exceptions between builds can be a strong indicator of bugs fixed.



CORRELATION OF ERRORS WITH PREVIOUS OCCURRENCES

Having historical records of previous failures that are similar to the ones currently being investigated is another invaluable tool that can greatly reduce time to fix errors. When code exceptions remain unchanged across the many microservices affected by a bug, developers can take insights from previous occurrences of a similar bug. This is rarely taken advantage of without a centralized, cross-build service like Rollbar that can collect code exceptions, group them, and bubble up this information at the right time.



CHECKING THE EFFECTIVENESS OF THE QA PROCESS

Through tracking of code exceptions and error monitoring across all phases of the SDLC, benefits of stability and bug fixing are accomplished. In addition, insights can be obtained that show the effectiveness that many phases of the SDLC can identify and fix specific issues in functionality.

By observing which exceptions are triggered early in the SDLC, but leak through each phase of the funnel unnoticed and ultimately reach production, improved test coverage or engineering process can be measured for better quality before software changes reach the deployment phase.

A FEW REAL-WORLD EXAMPLES

CUTTING-EDGE TECH MEETS THE FINANCE WORLD

For a trusted company thriving in the FinTech space, this major enterprise organization is an outlier. Their practices, tech stack, and team resemble those of a small and lean startup rather than a financial institution. This has allowed them to deliver vast amounts of value to their users and disrupt the spaces of loans and financial management.

Observability and reliability practices have always been important for their processes, but not a big driving force. They have chosen not to hand-hold their engineers and instead build a relationship of trust with each individual and team. Engineers have a high degree of ownership through the DEV and QA environments.

"WE HAVE BASICALLY EVERYTHING IN ROLLBAR TO SOME EXTENT." - *MARCUS YARBRO, Senior Software Engineer*

When it comes to stability and resembling production closely, the staging phase and environment are key. Aside from certain exceptions around smoke or regression testing, their team strives for a high level of stability in staging.

To support the extraordinary need for stability and quality required in the financial space, a new environment was created. This shadow environment matches production to a higher degree than most staging environments in tech companies. Scale, data, and infrastructure are matched one to one. Stress tests can push the shadow environment to handle up to 12 times the load of production.

When it comes to DEV, local environments aren't an option for developers, so this financial company relies on an internal PaaS (Platform as a Service) that allows individuals to spin up a full ephemeral environment when needed.

This process and set of environments allow their team to release code to production around 90 times a day. Before adopting Rollbar, the team relied on an ELK setup (Elasticsearch + Logstash + Kibana) for tracking and reporting exceptions. The pace of exceptions and the real-time need to report on these during their CI builds weren't met by that setup, regardless of the investment and manpower dedicated to keeping it responsive.

Since replacing the legacy setup, Rollbar has been adopted across the system and in every environment of the SDLC past DEV. They have adopted the practice of "fingerprinting," and using Rollbar and an additional transaction identifier with each exception, grouping, and correlation of exception across multiple microservices has been of great value.

By integrating Rollbar with JIRA, tickets are created for every single error collected across all environments. Being able to "link" or "de-duplicate" bug reports by tracking the exception of origin that caused them in both the same build as their previous occurrences in unrelated builds has delivered huge gains.

The team hasn't achieved building correlations between test failures and their original exceptions yet but sees this as a natural progression to their current observability practices. Measuring and reporting the effectiveness of QA and staging phases by tracking exceptions would be a natural next step after that.

MAJOR AUTO INSURER MEASURES DEFECT ESCAPE RATE

For one of the largest auto insurance companies in the USA, their SDLC is what most technical leaders would label as reliable and thorough. With a business that communicates trust and maturity, the cost of bugs reaching end users has a considerable psychological component, aside from just the financial impact. This drives an average release cycle of 3 weeks, with hotfix releases within the day.

One of their top priorities when it comes to issues caught along the development, testing, and release process, is to direct the issue most effectively to the developer responsible (and therefore most qualified) for its resolution. Originally through software built in-house, they correlated logging and codebase characteristics to create JIRA tickets for each "risk" discovered and effectively route it to the right person. This process was surprisingly effective and worked the majority of times.

With the goal of maturity, the set environments include sandboxes for development, an environment used for integration and E2E (end to end testing), an environment used exclusively for UAT (user acceptance testing), a load testing environment, an internal training environment, and ultimately, production. Large applications can have upwards of 10 non-production environments, with a minimum of 5. All of them are long-lasting and well-managed.

> ***"THE MORE WE CAN CATCH IN STAGING AND SHADOW, THE LESS WE HAVE TO DEAL WITH IN PRODUCTION."*** - *MARCUS YARBRO, Senior Software Engineer*

A practice instituted at the cost of some valuable Rollbar features is to group "risks" or bug occurrences across all environments to obtain insights and data for a single issue across the full SDLC. This has allowed the engineering leaders to realize that the impact of an error on QA or UAT is unlikely to be representative of its impact in production. Thanks to this insight, they treat every single risk discovered through error monitoring as a top priority, with its own bug tracking, analysis, and risk assessment.

A valuable addition to their error monitoring is the concept of "perpetrator" and "victim" when it comes to exceptions. Because of the many layers of abstraction that an enterprise stack like theirs is composed of (interfaces, middlewares, framework abstractions), a single exception thrown will cascade into many. Being able to correlate these across the full stack and automatically group them has considerably reduced triaging times.

This large enterprise is no stranger to bringing observability practices into earlier phases of the SDLC. Aside from error monitoring through Rollbar, other practices like logging and telemetry are used across most environments and important insights are obtained from them to assess risks and improve bug reports. They obtains great benefits from reducing resolution times thanks to these. The quality assurance organization praises the value of the current error monitoring tooling and the constant tracking of the effectiveness of the testing and QA phases of the SDLC.

With improved tooling and features specifically built for monitoring errors and exceptions in earlier phases of the SDLC, they would greatly benefit from better deduplication of bug reports both in the same CI build, as well as being able to correlate errors from different builds, effectively providing details about the recurrence of an issue to development, and metrics about this, to management.

Finally, their leadership is intensely focused on Defect Escape Rate. Through tracking and monitoring of errors throughout the stack and through all phases of the software development lifecycle, they can assess the rate that bugs pass QA, UAT, and even load testing unattended, eventually getting to production. Through closely monitoring and improving this metric, they can effectively strive for delivering a better and more reliable experience to their users every day.

CONCLUSION

Through the implementation of mature operations and monitoring practices using Rollbar in all phases of the Software Development Life Cycle, small and large organizations are already improving environmental stability, detecting issues earlier, easing resolution, and improving the quality of their processes. Rollbar is geared to become a key innovator in this space, building dedicated features to better support these practices and enhance the benefits they deliver.

Engineering leaders of organizations relying on multiple environments, who are looking to improve the quality of their software and reduce the time to release new changes, should strongly consider error monitoring through Rollbar to obtain these benefits.

NEXT STEPS

If you're interested in adopting the proposed practices and getting some of the benefits described above, here are some next steps to take:

1. Sign up and integrate Rollbar into your system - **rollbar.com/signup**
2. Enable Rollbar and track exceptions for all environments in the SDLC
3. Start reporting environment details to Rollbar through the built-in properties
4. Set up the source control integration with your CVS of choice
5. Set up fingerprinting of occurrences across different microservices
6. Set up the issue tracking integration to add Rollbar issues to your SDLC workflow
7. Build dedicated reports to improve observability based on the new data collected

CONTACT US

rollbar.com
team@rollbar.com
1.888.568.3350