Rollbar

Whitepaper

Why You Need Error Grouping: From Alerts to Action

How to Sift Through Crash Alerts, Pinpoint the Right Errors, and Start Debugging Better

Introduction

Start Debugging Better

You've always known debugging is important. But did you know that it can take up to half of your developers' time? According to a 2019 report published in ACM Queue, as well as a classic University of Cambridge study, developers spend 35–50% of their time debugging.

Debugging, testing, and verification combined form up to 75% of the cost of software development projects, totaling more than $100 billion annually worldwide.

Clearly, anything we can do to make debugging more efficient will lead to major cost savings—especially since when developers are trying to debug, there's never just one error. A single problem in the code can easily overwhelm developers with hundreds or thousands of alerts, sometimes even more, depending on the number of end users affected. In fact, if the code is broken, it's rare for there to be just one error, since different parts of the application or service will be crashing and raising alerts in turn, creating thousands of potential error messages.

With all this in mind, how can you help your development team work more efficiently and improve the software development experience? In this post, we'll answer that question. We'll also discuss tools you can use to automate and streamline debugging to find and resolve problems faster.

Developers have to sift through all these alerts to find the ones that really matter and pinpoint the root cause of the errors.

This leads to three main problems:

1 Wasted Time

First, developers don't know which errors are related to the same cause, so they have to triage them individually and determine which ones to spend time resolving. Second, there's a chance they'll end up fixing irrelevant errors and delivering product updates that don't actually resolve the problem. And these irrelevant fixes may introduce new errors, starting the cycle of wasted time all over again.

2 Noise and Alert Fatigue

The development environment is already filled with noise coming in from multiple angles: analytics, APIs, databases, and dependencies. With too many error alerts coming in on top of this, the development team may have a tough time prioritizing tasks, potentially leading to burn out. Plus, when constantly faced with numerous alerts, developers may actually start to ignore them, treating them as "business as usual." That means that they may not react appropriately when faced with a truly high-priority issue.

3 Dev Team Frustration

When developers have to sift through alerts, they are less likely to be satisfied with their day-to-day work. They'd prefer to be creating something new and coding—not "wasting time" wading through errors. If they're constantly reacting to alerts, they have less time for their primary work.

How Grouping Helps— And When It Can't

Errors are inevitable—there's no such thing as perfect software. Whether in development or production, errors can occur due to a wide range of reasons that fall within five main categories:

1. **User Related**

   Failing to anticipate user action or behavior

2. **Code Quality and Validation**

   Basic syntax or logic errors within the code; not validating data for completeness and correctness; failing to anticipate system behavior at scale

3. **Software Dependencies**

   Unavailable libraries or other resources; databases offline

4. **Hardware Dependencies**

   System errors or unavailable external systems; Internet connection problems

5. **Security**

   New publicly known information-security vulnerabilities and exposures affecting your code or dependencies

Error signals don't always contain much meaningful data and context to help developers pinpoint the source of the problem. What developers really want—and need—to get their work done efficiently is to be able to deal with errors as fast as possible.

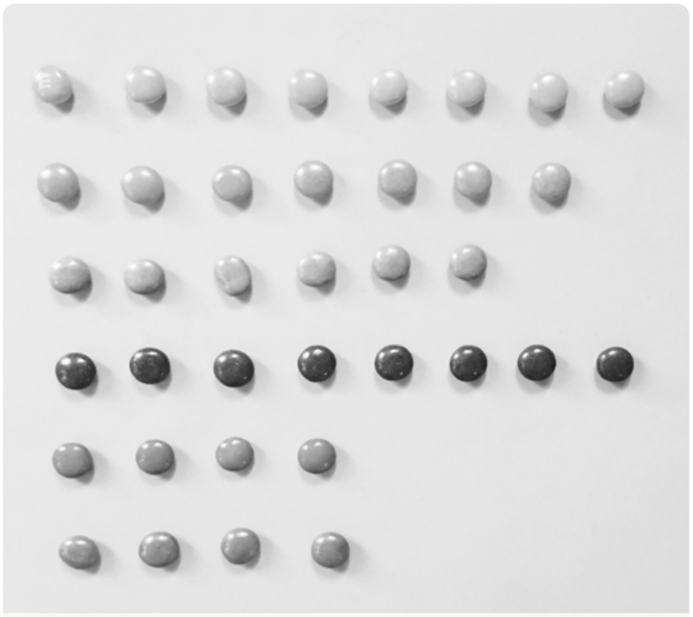
To accomplish this, they need:

- Trustworthy signals to let them know when a bug truly needs fixing
- An accurate list of errors to help them plan and optimize their debugging activities
- A way to separate new errors and occurrences from known bugs to help them triage and prioritize

Unfortunately, what developers get instead is a deluge of unsorted error signals (alerts) based on crashes that are caused by an underlying error. They can't easily identify which alerts have the same root cause (and can therefore be solved with the same fix) and which ones are completely separate (and will eventually require a fix of their own).

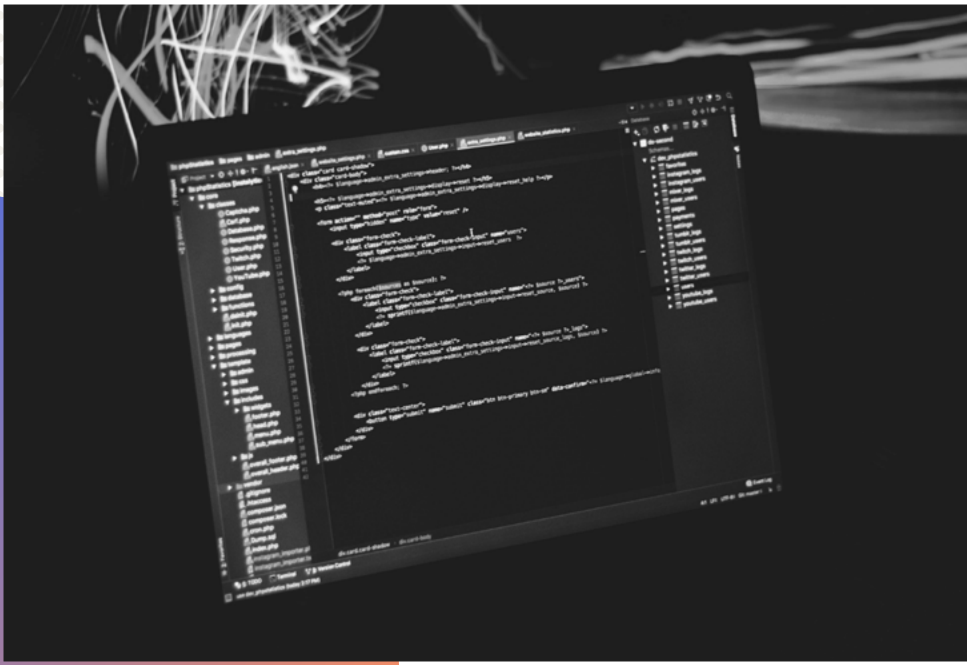
Imagine you have a bag of 100 M&M's. You need to figure out as quickly as possible how many different colors there are in the bag and how many M&M's there are of each color. How much time would you save if someone simply told you the answer? Maybe you'd even appreciate more advanced insights, such as the odds of choosing a brown one (somewhere around 12%). That's exactly the kind of system developers need—one that sifts through all those error reports to determine which ones are related, so they don't have to do it by hand.

The most efficient solution here is to group errors. It sounds intuitive, yet too few developers are using this strategy. With grouping, errors with the same root cause are sorted together into an organized and classified list to help developers debug more effectively.



There are many advantages to grouping errors, including:

- Saving time, with no more searching through all those alerts and logs; the list of errors is cleared up and organized
- Receiving a more useful error context to help pinpoint the cause while weeding out negligible errors as "noise"
- Helping you determine the priority of each error so you can optimize dev team resources
- Ensuring that the single right fix will clear many errors off your plate, since they all stem from the same cause to begin with
- Opening the door to automation, since with trustworthy alerts, you can automate feature flags, create tickets, and introduce sequential workflows in many other ways (such as halting or rolling back a progressive deployment)



The Limits of Grouping Tools

Grouping can help produce better error signals, but traditional techniques don't go far enough. Until now, if a development team wanted to integrate error grouping into its work process, it would rely on one or more of a few traditional manual techniques. For example, the team might aggregate all its logs and then create filters and queries that are customized to its data set to cut through the noise.

Tools have been available for a while to help with grouping errors, yet most of them rely on a "hard-coded" grouping approach. They identify bugs based on recognized, manually-curated patterns, meaning that an individual is writing the rules. By definition, this will only include existing error types that have already been identified.

These solutions are analogous to antivirus engines with hard-coded signatures. They're great at spotting malicious patterns that are already out there, but they're lousy at "zero-day" attacks (brand-new threats that nobody has seen before). They also still require a substantial amount of investigative and set-up work, which, of course, takes time and diverts team members who could be coding productively.

When it comes to reducing noise, the variety of potential errors is far too large to be maintained manually—and with ever-changing coding frameworks, the complexity is only going to increase. In fact, this type of hard-coded grouping logic may actually make the problem worse because it creates:

- **False positives:** The tool fails to identify multiple errors stemming from the same issue, so it won't weed out all of the errors.
- **False negatives:** The tool groups errors that are actually unrelated. This means separate problems won't get flagged separately and may fall between the cracks.

Either way, users won't get a stack-ranked error list they can rely on. And without reliable and accurate grouping from a tool they trust, users will often choose to group errors manually or keep dealing with the flood of errors in some other way. So while error grouping holds a lot of potential, today's fast-paced development environments need a more flexible model.

Solution: Grouping— with Continuous Learning

Today, many developers are still opting to handle grouping manually. In fact, a 2017 academic study on debugging strategies among Windows and Android developers found that by and large, developers don't believe fault-localization tools can help find tough bugs or pinpoint the source of errors. They also think that manual debugging techniques, like stack tracing, are sufficient. That's probably because many tools that promise to make debugging more efficient are both inflexible and inaccurate (as we've just seen).

Also, when discussing the possibility of switching to better fault-localization techniques, the developers in the study resisted change due to the learning curve, feared the new technique wouldn't work well with existing environments, and doubted it could do what it claimed to do.

Clearly, to win back developers' trust, you need a tool that not only provides continuous learning for more intelligent error grouping, but is also easy to learn and use. And it must provide a level of accuracy and reliability that developers can't achieve otherwise.

For today's constantly changing development environments, the best solution is a continuously learning grouping engine that can:

- ✓ Detect new and unique error types
- ✓ Reduce noise and alert fatigue with more accurate error signals
- ✓ Prioritize what to fix by grouping root causes and allowing users to see error context (functions impacted, what browser/IPs are affected, and other factors)

Also, a tool with a built-in continuous learning engine makes development teams more productive because it lets them respond to critical errors faster. They spend less time digging through logs or using other manual methods to find the root causes of errors and then attempting to fix them.

Rollbar's Automation—Grade Grouping

Rollbar is a modern, continuous code-improvement tool designed to streamline the software development experience. This is the solution developers have been waiting for.

It uses machine learning to intelligently group bugs so developers can find and fix problems faster—and get back to creating code.

Rollbar analyzes both exceptions and log messages. Exceptions are clustered by platform, environment, error class, and stack trace information. Message-type items are grouped by message text (stripping out number-like and date-like portions). Rollbar's algorithm uses three guiding principles, based on developers' needs:

1. Occurrences with the same root cause are combined (this eliminates false positives).
2. Occurrences with different root causes are kept separate (this eliminates false negatives).
3. Performance is maximized by learning from millions of errors and adapting the algorithm accordingly.

These principles ensure that noise is minimized—and so are missed bugs. It also makes triaging easier for developers because duplicates are cleared from the error list, letting the real issues emerge more clearly. By eliminating a big chunk of the debugging process, developers will be able to respond to errors faster—with less work.

The Automation-Grade Grouping that powers Rollbar is continuously updated and improved, so you'll see better recognition as Rollbar becomes familiar with your code. You can also create custom fingerprint rules to override the default behavior and reach the level of grouping accuracy you want even faster. Rollbar's grouping behavior adapts to your projects and environments, as needed, in ways that standard grouping algorithms can't. The ultimate result is better code and a better quality product.

Conclusion

The goal of software development is to get a reliable product into customers' hands. Companies spend billions of dollars a year on debugging throughout the development and production environments to reduce application crashes and increase customer satisfaction.

Rollbar's continuous code-improvement solution works just as hard as your development team does. With Automation-Grade Grouping, Rollbar finds bugs before users do and helps you fix them before they escalate, saving time and money while improving the software development experience, code quality, and your release cycle.

To start debugging better with Rollbar, request a demo or try it out for free.

Rollbar

Rollbar is the leading continuous code improvement platform that enables developers to proactively discover and resolve issues in their code. With Rollbar, developers focus on deploying better software faster, knowing they can quickly recover from critical errors as they happen. Learn more at Rollbar.com